METHOD OF DETERMINING DATA PLACEMENT FOR DISTRIBUTED STORAGE SYSTEM

Related Applications

This application is related to U.S. Application Nos. (Attorney Docket Nos. 200311959-1, 200311960-1, 200311961-1, and 200312488-1), filed on (the same day as this application), the contents of which are hereby incorporated by reference.

Field of the Invention

The present invention relates to the field of data storage. More particularly, the present invention relates to the field of data storage where data is placed onto nodes of a distributed storage system.

Background of the Invention

A distributed storage system includes nodes coupled by network links. The nodes store data objects, which are accessed by clients. By storing replicas of the data objects on a local node or a nearby node, a client can access the data objects in a relatively short time. An example of a distributed storage system is the Internet. According to one use, Internet users access web pages from web sites. By maintaining replicas on nodes near groups of the Internet users, access time for the Internet users is improved and network traffic is reduced.

Replicas of data objects are placed onto nodes of a distributed storage system using a data placement heuristic. The data placement heuristic attempts to find a near optimal solution for placing the replicas onto the nodes but does so without an assurance that the near optimal solution will be found. Broadly, data placement heuristics can be categorized as caching techniques or replication techniques. A node employing a caching technique keeps replicas of data objects accessed by the node. Variations of the caching technique include LRU (least recently used) caching and FIFO (first in first out) caching. A node employing LRU caching adds a new data object upon access by the node. To make room for the new data object, the node discards a data object that was most recently accessed at a time earlier than other data objects stored on the node. A node employing FIFO caching also adds a new data object upon access by the node but it discards a data object based upon load time rather than access time.

The replication techniques seek to make placement decisions about replicas of data objects typically in a more centralized manner than the caching techniques. For example, in a completely centralized replication technique, a single node of the distributed storage system decides where to place replicas of data objects for all data objects and nodes in the distributed storage system.

Currently, a system designer or system administrator seeking to deploy a placement heuristic in order to place replicas of data objects within a distributed storage system will choose a data placement heuristic in an ad-hoc manner. That is, the system designer or administrator will choose a particular data placement heuristic based upon intuition and past experience but without assurance that the data placement heuristic will perform adequately.

What is needed is a method of determining data placement for a distributed storage system.

Summary of the Invention

The present invention comprises a method of determining data placement for a distributed storage system. An embodiment of the method of determining the data placement begins with a selection of a heuristic class for placing data objects onto nodes of the distributed storage system. The heuristic class meets a performance requirement and provides a replication cost that is within an allowable limit of a minimum replication cost. The method continues with an instantiation of a data placement heuristic selected from a range of data placement heuristics according to the heuristic class.

According to one embodiment, the method concludes with the instantiation of the data placement heuristic. According to another embodiment, the method concludes with an evaluation of a placement of the data objects onto the nodes made according to the data placement heuristic. According to another embodiment, the method continues by iteratively performing the selection of the heuristic class, the instantiation of the data placement heuristic, and the evaluation of the data placement heuristic.

According to one embodiment, the instantiation of the data placement heuristic comprises simulating the instantiation of the data placement heuristic. According to another embodiment, the instantiation of the data placement heuristic comprises instantiating the data placement heuristic on an actual distributed storage

system operating with an actual workload.

These and other aspects of the present invention are described in more detail herein.

Brief Description of the Drawings

The present invention is described with respect to particular exemplary embodiments thereof and reference is accordingly made to the drawings in which:

Figure 1 illustrates an embodiment of a distributed storage system of the present invention;

Figure 2 illustrates an embodiment of a method of selecting a heuristic class for data placement in a distributed storage system of the present invention as a flow chart;

Figure 3 provides a table of decision variables according to an embodiment of the method of selecting the heuristic class of the present invention;

Figure 4 provides a table of specified variables according to an embodiment of the method of selecting the heuristic class of the present invention;

Figure 5 provides a table of heuristic classes and heuristic properties which model the heuristic classes according to an embodiment of the method of selecting the heuristic class of the present invention;

Figure 6 illustrates an embodiment of a rounding algorithm of the present invention as a flow chart;

Figure 7 illustrates an embodiment of a method of instantiating a data placement heuristic of the present invention as a flow chart; and

Figure 8 illustrates an embodiment of a method of determining data placement of the present invention as a block diagram.

Detailed Description of a Preferred Embodiment

Data is often accessed from geographically diverse locations. By placing a replica or replicas of data near a user or users, data access latencies can be improved. An embodiment for accomplishing the improved data access comprises a geographically distributed data repository. The geographically distributed data repository comprises a service that provides a storage infrastructure accessible from geographically diverse locations while meeting one or more performance requirements such as data access latency or time to update replicas. Embodiments of

the geographically distributed data repository include a personal data repository and remote office repositories.

The personal data repository provides an individual with an ability to access the personal data repository with a range of devices (e.g., a laptop computer, PDA, or cell phone) and from geographically diverse locations (e.g., from New York on Monday and Seattle on Tuesday). When the individual opts for the personal data repository, data storage for the individual becomes a service rather than hardware, eliminating the need to physically purchase the hardware and eliminating the need to maintain it. For an individual who travels frequently, it would be especially beneficial in its elimination of the need to carry the hardware from place to place.

The provider of the personal data repository guarantees the performance requirements to the individual. In an embodiment of the personal data repository, the performance requirements comprise guaranteeing data access latency to files within a period of time, for example 1 sec. In another embodiment of the personal data repository, the performance requirements comprise a data bandwidth guarantee. For example, the data bandwidth guarantee could be guaranteeing that VGA quality video will be delivered without glitches. In another embodiment of the personal data repository, the performance requirements comprise an availability guarantee. For example, the availability guarantee could be guaranteeing that data will be available 99% of the time.

Other features envisioned for the personal data repository include data security, backup services, and retrieval services. The data security for the individual can be ensured by providing an access key to the individual. The backup and retrieval services could form an integral part of the personal data repository. The personal data repository also provides a convenient mechanism for the individual to share data with others, for example, by allowing the individual to maintain a personal web log. It is anticipated that the personal data repository would be available to the individual at a cost comparable to hardware based storage.

The remote office repositories provide employees with access to shared files. The performance requirements for the remote office repositories could be data access latency, data bandwidth, or guaranteeing that other employees would see changes to the shared files within an update time period. For example, the update time period could be 5 minutes. Other features envisioned for the remote office repositories

include the data security, backup services, and retrieval services of the personal data repository.

An exemplary embodiment of the remote office repositories comprises a system configured for a digital movie production studio. The system allows an employee to work on an animation scene from home using a laptop incapable of holding the animation scene by meeting certain performance requirements of data access latency and data bandwidth. Upon updating the animation scene, other employees of the digital movie production studio that have authorized access would be able to see the changes to the animation scene within the update time period.

The present invention addresses the performance requirements of geographically distributed data repositories while seeking to minimize a replication cost. According to an aspect, the present invention comprises a method of selecting a heuristic class for data placement from a set of heuristic classes. Each of the heuristic classes comprises a method of data placement. The method of selecting the heuristic class seeks to minimize the replication cost by selecting the heuristic class that provides a low replication cost while meeting the performance requirement.

Each of the heuristic classes represents a range of data placement heuristics. A heuristic comprises a method employed by a computer that uses an approximation technique to attempt to find a near optimal solution but without an assurance that the approximation technique will find a near optimal solution. Heuristics work well at finding the quasi optimum solution provided that a problem definition for a particular problem falls within a range of problem definitions appropriate for a selected heuristic.

One skilled in the art will recognize that the term "heuristic" can be employed narrowly to define a search technique that does not provide a result which can be compared to a theoretical best result or it can be employed more broadly to include approximation algorithms which provide a result which can be compared to a theoretical best result. In the context of the present invention, the term "heuristic" is used in the broad sense, which includes the approximation algorithms. Thus, the term "approximation technique" should be read broadly to refer to both heuristics and approximation algorithms.

An embodiment of the method of selecting the heuristic class comprises solving a general integer program to determine a general lower bound for the replication cost, solving a specific integer program to determine a specific lower

bound for the replication cost for a heuristic class, and comparing the general lower bound to the specific lower bound. In this embodiment, the method selects the heuristic class if the specific lower bound is within an allowable limit of the general lower bound.

Another embodiment of the method of selecting the heuristic class comprises solving first and second specific integer programs for each of first and second heuristic classes to determine first and second specific lower bounds for the replication cost for each of the first and second heuristic classes. In this embodiment, the method selects the first or second heuristic class depending upon a lower of the first or second specific lower bounds, respectively.

A further embodiment of the method of selecting the heuristic class comprises solving the general integer program and the first and second specific integer programs. In this embodiment, the method selects the first or second heuristic class depending upon a lower of the first or second specific lower bounds, respectively, if the lower of the first or second specific lower bounds is within the allowable lime of the general lower bound.

The general and specific integer programs for determining the general and specific lower bounds for the replication costs are NP-hard. (The term "NP-hard" means that there is no known algorithm that can solve the problem within any feasible time period, unless the problem size is small.) Thus, an exact solution is only available for a small system. According to an aspect, the present invention comprises a method of determining a lower bound for the replication cost where the lower bound comprises the general lower bound (for any conceivable heuristic) or the specific lower bound (for a specific class of heuristics). An embodiment of the method of determining the lower bound comprises solving an integer program using a linear relaxation of binary variables to determine a lower limit on the lower bound and performing a rounding algorithm until all of the binary variables have binary values, which determines an upper limit on an error for the lower bound.

According to another aspect, the present invention comprises a method of instantiating a data placement heuristic using an input of a plurality of heuristic parameters. In an embodiment of the method of instantiating the data placement heuristic, a node of a distributed storage system receives the heuristic parameters and runs an algorithm, which places data objects on nodes that are within a designated set of nodes. In another embodiment of the method of instantiating the data placement

heuristic, a system simulating a node of a distributed storage system receives the heuristic parameters and runs the algorithm, which simulates placing data objects on nodes that are within a node scope.

According to a further aspect, the present invention comprises a method of determining data placement for the distributed storage system. In an embodiment of the method of determining the data placement, a system implementing the method selects a heuristic class and instantiates a data placement heuristic using the heuristic class. Another embodiment comprises selecting the heuristic class, instantiating the data placement heuristic, and evaluating a resulting data placement. In one embodiment, the step of evaluating the resulting data placement comprises simulating implementation of the data placement on a system experiencing a workload. In another embodiment, the step of evaluating the resulting data placement comprises simulating implementation of the data placement on at least two different system configurations experiencing a workload in order to determine which of the system configurations provides better efficiency or better performance. In a further embodiment, the step of evaluating the resulting data placement comprises implementing the data placement on a distributed storage system experiencing an actual workload.

An embodiment of a distributed storage system of the present invention is illustrated schematically in figure 1. The distributed storage system 100 comprises first through fourth nodes, 102..108, coupled by network links 110. Clients 112 coupled to the first through fourth nodes, 102..108, access data objects within the distributed storage system 100. Additional network links 114 couple the first through fourth storage nodes, 102..108, to additional nodes 116. Each of the first through fourth nodes, 102..108, and the additional nodes 116 comprises a storage media for storing the data objects. Preferably, the storage media comprises one or more disks. Alternatively, the storage media comprises some other storage media such as a tape. A data placement heuristic of the present invention places replicas of the data objects onto the first through fourth nodes, 102..108, and the additional nodes 116.

Mathematically, the first through fifth nodes, 102..108, and the additional nodes 116 are discussed as n nodes where $n \in \{1, 2, 3, \ldots N\}$, where N is the number of nodes. Also, the data objects are discussed mathematically as k data objects where $k \in \{1, 2, 3, \ldots K\}$, where K is the number of data objects.

While the distributed storage system 100 is depicted with the *n* nodes, it will be readily apparent to one skilled in the art that the methods of the present invention apply to the distributed storage system 100 having as few as two of the nodes.

An embodiment of the method of selecting the heuristic class for the data placement of the present invention is illustrated as a flow chart in figure 2. The method of selecting the heuristic class 200 begins in a first step 202 of receiving inputs. The inputs comprise a system configuration, a workload, and a performance requirement. The system configuration represents the distributed storage system 100. The workload represents users requesting data objects from the *n* nodes. The performance requirement comprises a bi-modal performance metric, which comprises a criterion and a ratio of successful attempts to total attempts. According to one embodiment, the performance requirement comprises a data access latency specified as a period of time for fulfilling a ratio of successful data accesses to total data accesses. An exemplary data access latency comprises data access within 250 ms for 99% of data access requests. According to another embodiment, the performance requirement comprises a data access bandwidth, a data update time, an availability, or an average data access latency.

The method of selecting the heuristic class 200 continues in a second step 204 of forming integer programs. According to an embodiment, the integer programs comprise the general integer program and the specific integer program. The general integer program models data placement irrespective of a data placement heuristic used to place the data objects. Solving the general integer program provides the general lower bound for the replication cost, which provides a reference for evaluating the heuristic class. The specific integer program models the heuristic class. The specific integer program comprises the general integer program plus one or more additional constraints.

The general and specific integer programs model the *n* nodes storing replicas of the *k* data objects. Each of the *n* nodes has a demand for some of the *k* data objects, which are requests from one or more users on the node. The one or more users can be one or more of the clients 112 or the user can be the node itself. The replicas of the *k* data objects can be created on or removed from any of the *n* nodes. These changes occur at the beginning of an evaluation interval. The evaluation interval comprises a time period between executions of the data placement heuristic for one of the *n* nodes. For example, a caching heuristic which is run upon the first node 102 for every access

of any of the k data objects from the first node 102 has an evaluation interval of every access. In contrast, a complex centralized placement heuristic which is run once a day has an evaluation interval of 24 hours.

According to an embodiment, an evaluation interval period Δ, i.e., a unit of time, is used to model the evaluation intervals even for the caching heuristic. An execution of a data placement heuristic comprises a set of all of the evaluation intervals modeled by the general and specific integer programs. Mathematically, the evaluation intervals are discussed herein as i evaluation intervals where $i \in \{1, 2, 3, \ldots I\}$, where I is the number of evaluation intervals. A selection of the evaluation interval period Δ should reflect the heuristic class that is modeled by the specific integer program for at least two reasons. First, as the evaluation interval period Δ decreases, a total number of the i evaluation intervals increases. This increases a number of computations for solving the general and specific integer programs and, consequently, increases a solution time. Second, as the evaluation interval period Δ decreases, the specific lower bound theoretically converges to a lowest possible value. The lowest possible value may be far lower than the replication cost for an actual implementation of a data placement heuristic.

According to an embodiment, the evaluation interval period Δ is selected in one of two ways depending upon the heuristic class that is being modeled. For heuristic classes that perform placements every P units of time, the evaluation interval period Δ is given by $\Delta = P_{min}/2$, where P_{min} is a smallest evaluation interval period on any of the n nodes for the execution of a data placement heuristic. For heuristic classes that perform placements after every access on an nth node, the evaluation interval period Δ is a minimum time between any two accesses of any of the n nodes.

The integer programs include decision variables and specified variables. According to an embodiment, the decision variables comprise variables selected from variables listed in Table 1, which is provided as figure 3. According to an embodiment, the specified variables comprise variables selected from variables listed in Table 2, which is provided as figure 4.

The general integer program comprises an objective of minimizing the replication cost. According to an embodiment, the objective of minimizing the replication cost is given as follows.

$$\sum_{i \in I} \sum_{n \in N} \sum_{k \in K} (\alpha \cdot store_{nik} + \beta \cdot create_{nik})$$

According to an embodiment, the general integer program further comprises general constraints. A first general constraint imposes the performance requirement on each of the nodes by constraining the decision variables so that the ratio of the successful accesses to the total accesses is at least a specified ratio T_{qos}. According to an embodiment, the first general constraint is given as follows.

$$\frac{\sum_{i\in I}\sum_{k\in K} read_{nik}\cdot covered_{nik}}{\sum_{i\in I}\sum_{k\in K} read_{nik}} \geq T_{qos} \qquad \forall n$$

A second general constraint imposes a condition that, if a replica of a kth data object is created on an nth node in an ith evaluation interval, the replica exists for the ith evaluation interval. According to an embodiment, the second general constraint is given as follows.

$$create_{nik} \geq store_{nik} - store_{n,i-1,k} \qquad \forall n,i,k$$

A third general constraint imposes a condition that initially no replicas exist in the distributed storage system. According to an embodiment, the third general constraint is given as follows.

$$store_{n,-1,k} = 0 \qquad \forall n,k$$

In an alternative embodiment, the third general constraint is modified to account for an initial placement of replicas of the k data objects on the n nodes.

A fourth general constraint imposes the condition that the nth node can access an mth node within a latency threshold T_{lat}. According to an embodiment, the fourth general constraint is given as follows.

$$covered_{nik} \leq \sum_{m\in N} dist_{nm}\cdot store_{mik} \qquad \forall n,i,k$$

A fifth general constraint imposes a condition that variables $store_{nik}$, $covered_{nik}$, and $create_{nik}$ are binary variables. According to an embodiment, the fifth general constraint is given as follows.

$$store_{nik}, covered_{nik}, create_{nik} \in \{0,1\} \qquad \forall n,i,k$$

According to an alternative embodiment, a penalty term is added to the objective of minimizing the replication cost. The penalty term reflects a secondary objective of minimizing data access latencies $latency_{nm}$ which exceed the latency threshold T_{lat}. According to an embodiment, the penalty term is given as follows.

$$\gamma\sum_{i\in I}\sum_{n\in N}\sum_{k\in K}(read_{nik}\cdot(1-covered_{nik})\cdot\sum_{m\in N}(latency_{nm}-T_{lat})\cdot route_{nmik})$$

According to an alternative embodiment, a first additional cost term is added to the objective of minimizing the replication cost. The first additional term captures a cost of writes in the distributed storage system. According to an embodiment, the first additional cost term is given as follows.

$$\delta \sum_{i \in I} \sum_{n \in N} \sum_{k \in K} (write_{nik} \cdot \sum_{m \in N} store_{mik})$$

According to an alternative embodiment, a second additional cost term is added to the objective of minimizing the replication cost. The second additional cost term reflects a cost of enabling a node to run a data placement heuristic and to store replicas of the k data objects. According to an embodiment, the second additional cost term is given as follows.

$$\zeta \cdot \sum_{n \in N} open_n$$

According to the alternative embodiment which includes the second additional cost term, additional general constraints are added to the general constraints. The additional general constraints impose conditions that an enablement variable $open_n$ is a binary variable and that the nth node must be enabled in order to store the k data objects on it. According to an embodiment, the additional general constraints are given as follows.

$$open_n \in \{0,1\} \qquad \forall n$$

$$open_n \geq store_{nik} \qquad \forall n,i,k$$

An embodiment of the specific integer programs adds one or more supplemental constraints to the general constraints of the general integer program. According to an embodiment, the supplemental constraints comprise constraints chosen from a group comprising a storage constraint, a replica constraint, a routing knowledge constraint, an activity history constraint, and a reactive placement constraint.

The storage constraint reflects a heuristic property that a fixed amount of storage is used throughout an execution of a data placement heuristic. For example, caching heuristics exhibit the heuristic property of using the fixed amount of storage. Thus, if the first integer program models a caching heuristic it would include the storage constraint. A global storage constraint imposes a condition of a fixed amount of storage for all of the n nodes and over all of the i intervals. According to an embodiment, the global storage constraint is given as follows.

$$\sum_{k \in K} store_{nik} = \sum_{k \in K} store_{0,0,k} \qquad \forall n,i$$

A local storage constraint imposes a condition of a fixed amount of storage over all of the i intervals and for each of the n nodes but it allows the fixed amount of storage to vary between the n nodes. According to an embodiment, the local storage constraint is given as follows.

$$\sum_{k \in K} store_{nik} = \sum_{k \in K} store_{n,0,k} \qquad \forall n,i$$

The replica constraint reflects a heuristic property that a fixed number of replicas for each of the k data objects are used throughout an execution of a data placement heuristic. Typically, centralized data placement heuristics use the fixed number of replicas. Thus, if the second integer program models a centralized data placement heuristic, it is likely to include the replica constraint. A first replica constraint imposes a condition of a fixed number of replicas for all of the k data objects and over all of the i intervals irrespective of demand for the k data objects. According to an embodiment, the first replica constraint is given as follows.

$$\sum_{n \in N} store_{nik} = \sum_{n \in N} store_{n,0,0} \qquad \forall i,k$$

A second replica constraint imposes a condition of a fixed number of replicas over all of the i intervals and for each of the k data objects but it allows the number of replicas to vary between the k data objects. According to an embodiment, the second replication constraint is given as follows.

$$\sum_{n \in N} store_{nik} = \sum_{n \in N} store_{n,0,k} \qquad \forall i,k$$

The routing knowledge constraints reflect a heuristic property of whether a node has knowledge of which others of the n nodes hold replicas of the k data objects. For example, if the nodes of a distributed storage system are using a caching heuristic, a node knows of the replicas stored on itself but has no knowledge of other replicas stored on other nodes. In such a scenario, if the node receives a request for a data object not stored on the node, the node requests the data object from an origin node. If the nodes of the distributed storage system are running a cooperative caching heuristic, a node knows of the replicas stored on nearby nodes or possibly all nodes. And if the distributed storage system is running a centralized heuristic, a node knows a closest node from which it can fetch a replica. According to an embodiment, the routing knowledge constraints employ a routing knowledge matrix $fetch_{nm}$ where

$fetch_{nm} = 1$ if an nth node knows of the replicas stored on an mth node and $fetch_{nm} = 0$ otherwise. According to the embodiment, the routing knowledge constraints are given as follows.

$$covered_{nik} \leq \sum_{m \in N} dist_{nm} \cdot store_{mik} \cdot fetch_{nm} \qquad \forall n, i, k$$

$$route_{nmik} - fetch_{nm} \leq 0 \qquad \forall n, m, i, k$$

An embodiment of the activity history constraint discussed below makes use of a sphere of knowledge matrix $know_{nm}$. When a data placement heuristic makes a placement decision for a node, the data placement heuristic takes into account activity at the node and potentially other nodes in the distributed storage system. For example, a caching heuristic makes placement decisions for a node based only on accesses to the node running the caching heuristic. Thus, when the caching heuristic is employed, the sphere of knowledge for a node is local. Or for example, a centralized heuristic makes placement decisions for all nodes in a distributed storage system based on accesses to all of the nodes. Thus, when the distributed storage system employs the centralized heuristic, the sphere of knowledge for a node is global. If a cooperative caching heuristic is employed, the sphere of knowledge for a node is regional. The sphere of knowledge matrix $know_{nm}$ indicates whether knowledge of accesses originating at an mth node is used to make placement decisions at an nth node. If so, $know_{nm} = 1$; and if not, $know_{nm} = 0$.

The activity history constraint reflects whether a data placement heuristic makes a placement decision based upon activity in one or more evaluation intervals. The one or more evaluation intervals include a current evaluation interval and previous evaluation intervals up to a specified number of intervals. If the current evaluation interval is used to make the placement decision, the placement decision is a forecast of a future event since the placement decision is made at the beginning of an evaluation interval. This is referred to as prefetching. If the previous evaluation interval is used to make the placement decision, the placement decision is based upon previous accesses for a data object.

The activity history constraint imposes the condition that a replica of a data object can be created if the data object has been created within the history and if the history is within a node's sphere of knowledge. For example, if a caching heuristic is employed, a replica of a data object is created if the data object was accessed within a single preceding interval by a node running the caching heuristic. Or for example, if a

centralized placement heuristic is employed and if the history is all intervals, a data placement heuristic considers the data objects accessed within the global sphere of knowledge. According to the embodiment of the activity history constraint, an activity history matrix $hist_{nik}$ indicates whether an nth node accessed a kth data object during or before an ith interval within a history considered by a data placement heuristic. If so, $hist_{nik} = 1$; if not, $hist_{nik} = 0$. According to the embodiment, the activity history constraint is given as follows.

$$create_{nik} \leq \sum_{m \in N} hist_{nik} \cdot know_{nm} \qquad \forall n, i, k$$

The reactive placement constraint reflects whether the prefetching is precluded. If the prefetching is precluded for a data placement heuristic, it is reactive heuristic. The reactive placement constraint imposes the condition that the activity history constraint cannot consider a current evaluation interval. For example, if a simple caching heuristic is employed, a replica of a data object is created if the data object was accessed within a single preceding interval by a node running the simple caching heuristic. Thus, for the simple caching heuristic, the prefetching is precluded. According to an embodiment, the reactive placement constraints are given as follows.

$$create_{nik} \leq \sum_{m \in N} hist_{n,i-1,k} \cdot know_{nm} \qquad \forall n, i, k$$

Solving the general integer program provides a general lower bound for the replication cost that applies to any data placement heuristic or algorithm. Solving the specific integer program provides the specific lower bound for the replication cost corresponding to a heuristic class for data placement. According to an embodiment, the heuristic class is described by heuristic properties, which comprise the supplemental constraints and other heuristic properties such as the sphere of knowledge matrix $know_{nm}$ and the activity history matrix $hist_{nik}$. According to an embodiment, some heuristic classes along with the heuristic properties which model them are listed in Table 3, which is provided as figure 5.

The method of selecting the heuristic class 200 (figure 2) continues in a second step 204 of solving the general and specific integer programs. According to an embodiment, solving each of the general and specific integer programs comprises an instantiation of the method of determining the lower bound. The method of determining the lower bound of the present invention is discussed above and more fully below. According to an alternative embodiment, the second step 202 of solving

the general and specific integer programs comprises an exact solution of the general or specific integer program. The alternative embodiment is less preferred because the exact solution is only available for a system configuration having a limited number of nodes.

The method of selecting the heuristic class 200 concludes in a third step 206 of selecting the heuristic class corresponding to the specific integer program if the specific lower bound for the replication cost of the heuristic class is within an allowable limit of the general lower bound. The allowable limit comprises a judgment made by an implementer depending upon such factors as the general lower bound (a lower general bound makes a larger allowable limit palatable), a cost of solving an additional specific integer program, and prior acceptable performance of the heuristic class modeled by the specific integer program. Typically, the implementer will be a system designer or system administrator who makes similar judgments as a matter of course in performing their tasks.

An alternative embodiment of the method of selecting the heuristic class comprises forming and solving the general integer program and a plurality of specific integer programs where each of the specific integer programs model a heuristic class. For example, a specific integer program could be formed for each of seven heuristic classes identified in Table 3 (figure 5). The alternative embodiment further comprises selecting the heuristic class which corresponds to the specific lower bound for the replication cost having a low value if the specific lower bound is within the allowable limit of the general lower bound.

An embodiment of the method of determining the lower bound of the present invention comprises solving an integer program using a linear relaxation of binary variables and performing a rounding algorithm. The integer program comprises the general integer program or the specific integer program. The binary variables comprise the decision variables $store_{nik}$ of the general integer program or of the specific integer program. Solving the integer program using the linear relaxation of the binary variables provides a lower limit for the lower bound. The rounding algorithm provides an upper limit for the lower bound.

An embodiment of the rounding algorithm of the present invention is illustrated as a flow chart in figures 6A and 6B. The rounding algorithm 600 begins in a first step 602 of receiving a cost, which has an initial value of the lower limit for the lower bound determined from the solution of the integer program using the linear

relaxation of the binary variables. The first step 602 further comprises receiving a performance, which has an initial value of the performance requirement. According to an embodiment of the rounding algorithm 600, the performance requirement comprises the specified ratio of successful accesses to total accesses T_{QoS}.

A second step 604 of the rounding algorithm 600 comprises determining whether any of the decision variables $store_{nik}$ have non-binary values. If not, the method ends because the linear relaxation of the binary variables has provided a binary result. However, this is unlikely. The decision variables $store_{nik}$ which have the non-binary values comprise a first subset.

The rounding algorithm continues in a third step 606, which comprises calculating a cost penalty, a performance increase, and a performance reward for each of the decision variables $store_{nik}$ within the first subset. According to an embodiment, the cost penalty *CostPenalty* is given by $CostPenalty = \alpha \cdot (1 - store_{nik})$, where α = the unit cost of storage. According to an embodiment, the performance increase *PerfIncrease* is given as follows.

$$PerfIncrease = \frac{(covered_{nik})_{binary} - (covered_{nik})_{nonbinary}}{\sum_{i \in I} \sum_{k \in K} read_{nik}}$$

Because the value of $covered_{nik}$ is constrained by the fourth general constraint above to a value no greater than one and because the non-binary value of $covered_{nik}$ may already have a value of one, the performance increase *PerfIncrease* may be found to be zero.

According to an embodiment, the performance reward *PerfReward* is given as follows.

$$PerfReward = \frac{(covered_{nik})_{binary}}{\sum_{i \in I} \sum_{k \in K} read_{nik}}$$

Unlike the performance increase *PerfIncrease*, the performance reward *PerfReward* will have a value greater than zero provided that the binary value of $covered_{nik}$ is one.

In a fourth step 608, the rounding algorithm picks the binary variable $store_{nik}$ from the subset which corresponds to a lowest ratio of the cost penalty *CostPenalty* to the performance reward *PerfReward* (i.e., a lowest value of *CostPenalty/PerfReward*) and removes it from the first subset. A fifth step 610 calculates the cost as a current cost value plus the cost penalty *CostPenalty* and calculates the performance as the current performance plus the performance increase *PerfIncrease*. A sixth step 612

determines whether any of the decision variables $store_{nik}$ remain in the first subset. If not, the method ends. Otherwise, the method continues.

In a seventh step 614, the rounding algorithm 600 determines which of the decision variables $store_{nik}$ within the first subset may be rounded down without violating the performance requirement. The decision variables $store_{nik}$ within the first subset which may be rounded down without violating the performance requirement comprise a second subset. An eighth step 616 determines whether the second subset includes any of the decision variables $store_{nik}$. If not, the rounding algorithm 600 returns to the third step 606. If so, the method continues.

In a ninth step 618, a cost reward *CostReward*, a performance penalty *PerfPenalty*, and the performance reward *PerfReward* are calculated for the binary variables $store_{nik}$ which remain in the second subset. According to an embodiment, the cost penalty *CostReward* is given by $CostReward = \alpha \cdot store_{nik}$, where α = the unit cost of storage. According to an embodiment, the performance increase *PerfPenalty* is given as follows.

$$PerfPenalty = \frac{(covered_{nik})_{nonbinary} - (covered_{nik})_{binary}}{\sum_{i \in I} \sum_{k \in K} read_{nik}}$$

A tenth step 620 determines whether the second subset contains one or more binary variables $store_{nik}$ with the performance reward *PerfReward* having a value of zero. If so, the one or more binary variables are rounded to zero and removed from the first subset. If not, an eleventh step 622 finds the binary variable $store_{nik}$ within the second subset with a highest ratio of the cost reward *CostReward* to the performance reward *PerfReward* (i.e., a highest value *CostReward/PerfReward*), rounds this binary variable to zero, and removes it from the first subset. A twelfth step 624 calculates the cost as a current cost value minus the cost reward *CostReward* and calculates the performance as a current performance minus the performance penalty *PerfPenalty*. An thirteenth step 626 determines whether any of the decision variables $store_{nik}$ remain in the first subset. If not, the method ends. Otherwise, the method continues by returning to the seventh step 314.

When the rounding algorithm 600 finds that no binary variables remain in the first subset, a fourteenth step 628 determines whether the integer program includes the storage constraint. If so, a fifteenth step 630 calculates the cost with storage maximized within an allowable storage. According to an embodiment, the storage

constraint comprises a global storage constraint. According to an embodiment which includes the global storage constraint, the cost calculated in the fifteenth step 630 is given as follows.

$$cost = cost_c + \alpha\sum_{i\in I}\sum_{n\in N}(c_{max} - \sum_{k\in K} store_{nik}) + \beta\sum_{n\in N}(c_{max} - c_n)$$

where $cost_c$ is the cost determined by the rounding algorithm prior to reaching the fiffourteenth step 630, where c_{max} is a maximum number of data objects stored on any of the n nodes during any of the i intervals, and where c_n is a maximum number of data objects stored on an nth node during any of the i intervals. According to another embodiment, the storage constraint comprises a nodal storage constraint. According to an embodiment which includes the nodal storage constraint, the cost calculated in the fifteenth step 630 is given as follows.

$$cost = cost_c + \alpha\sum_{i\in I}\sum_{n\in N}(c_n - \sum_{k\in K} store_{nik})$$

A sixteenth step 632 determines whether the integer program includes the replica constraint. If so, a seventeenth step 634 calculates the cost with replicas maximized within an allowable number of replicas. According to an embodiment, the replica constraint comprises a global replica constraint. According to an embodiment which includes the global replica constraint, the cost calculated in the seventeenth step 634 is given as follows.

$$cost = cost_c + \alpha\sum_{i\in I}\sum_{k\in K}(d_{max} - \sum_{n\in N} store_{nik}) + \beta\sum_{k\in K}(d_{max} - d_n)$$

where d_{max} is a maximum number of replicas of any of the k data objects stored during any of the i intervals and where d_n is a maximum number of replicas of a kth data object during any of the i intervals. According to an embodiment, the replica constraint comprises an object specific replica constraint. According to an embodiment which includes the object specific replica constraint, the cost calculated in the seventeenth step 634 is given as follows.

$$cost = cost_c + \alpha\sum_{i\in I}\sum_{k\in K}(d_n - \sum_{n\in N} store_{nik})$$

The method of determining the lower bound ends when the rounding algorithm 600 finds that no binary variables $store_{nik}$ remain in the subset and after considering whether the integer program includes the storage or replica constraint. If the integer program does not include the storage or replica constraint, the cost calculated in the fifth or twelfth step, 610 or 624, forms the upper limit on the lower

bound. If the integer program includes the storage constraint, the cost calculated in the fifteenth step 630 forms the upper limit on the lower bound. And if the integer program includes the replica constraint, the cost calculated in the seventeenth step 634 forms the upper limit on the lower bound.

According to an embodiment of the method of selecting the heuristic class, the lower limits comprise the lower bounds for the general and specific integer programs. In this embodiment, the upper limits provide a measure of confidence for the lower bounds. According to another embodiment of the method of selecting the heuristic class, the lower limit comprises the lower bound for the general integer program and the upper limit comprises the upper bound for the specific integer program. In this embodiment, the lower and upper bounds provide a worst case comparison between data placement irrespective of a data placement heuristic used to place the data and data placement according to a heuristic class modeled by the specific integer program.

According to an embodiment, the method of selecting the data placement heuristic of the present invention provides inputs for selecting heuristic parameters used in the method of instantiating the data placement heuristic of the present invention.

An embodiment of the method of instantiating the data placement heuristic comprises receiving heuristic parameters and running an algorithm to place data objects onto one or more nodes of a distributed storage system. According to an embodiment, the heuristic parameters comprise a cost function, a placement constraint, a metric scope, an approximation technique, and an evaluation interval. According to an alternative embodiment, the heuristic parameters comprise a plurality of placement constraints. According to another alternative embodiment, the heuristic parameters further comprise a routing knowledge parameter. According to another embodiment, the heuristic parameters further comprise an activity history parameter. By varying the heuristic parameters, the method of instantiating the data placement heuristic generates data placements corresponding to a wide range of data placements heuristics.

According to an embodiment, the heuristic parameters are defined with reference to the distributed storage system 100 (figure 1). The distributed storage system 100 comprises the first through fourth nodes, 102..108, and the additional nodes 116, represented mathematically as the n nodes where $n \in \{1, 2, 3, ..., N\}$. The distributed storage system further comprises the clients 112. The clients 112 are

represented mathematically as j clients where $j \in \{1, 2, 3, ..., J\}$. The data placement heuristics place the k data objects onto the n nodes where $k \in \{1, 2, 3, ..., K\}$. A jth client assigned to an nth node incurs a cost according to the cost function when accessing a kth data object. The distributed storage system 100 further comprises the network links and the additional network links, 110 and 114, which are represented mathematically as $l \in \{1, 2, 3, \ldots, L\}$.

The heuristic parameters are further defined according to problem definition constraints. A first problem definition constraint imposes a condition that each of the j clients sends a request for a kth data object to one and only one node. According to an embodiment, a request variable y_{jnk} indicates whether the ith client sends a request for a kth data object to an nth node. According to an embodiment, the first problem definition constraint is given as follows.

$$\sum_{n \in N} y_{jnk} = 1 \qquad \forall n, k$$

A second problem definition constraint imposes a condition that only an nth node that stores a kth data object can respond to a request for the kth data object. According to an embodiment, a storage variable $store_{nk}$ indicates whether an nth node stores a kth data object. According to an embodiment, the second problem definition constraint is given as follows.

$$y_{jnk} \leq store_{nk} \qquad \forall j, n, k$$

Third and fourth problem definition constraints impose conditions that the request variable y_{jnk} and the storage variable $store_{nk}$ comprise binary variables. According to an embodiment, the third and fourth problem definition constraints are given as follows.

$$y_{jnk}, store_{nk} \in \{0,1\} \qquad \forall j, n, k$$

The cost function comprises a client perceived performance or an infrastructure cost. A goal of the data placement heuristic comprises optimizing the cost function. According to an embodiment, the cost function comprises a sum of distances traversed by j clients accessing n nodes to retrieve k data objects. According to an embodiment, the sum of the distances is given as follows.

$$\sum_{j \in C} \sum_{n \in N} \sum_{k \in K} reads_{jk} \cdot dist_{jn} \cdot y_{jnk}$$

where a read variable $reads_{jk}$ indicates a rate of read accesses by a jth client reading a kth data object and where a distance variable $dist_{jn}$ indicates the distance between the

jth client and an nth node. According to an embodiment, the distance variable $dist_{jn}$ comprises a network latency between the jth client and the nth node. According to an alternative embodiment, the distance variable $dist_{jn}$ comprises a link cost between the jth client and the nth node.

According to an alternative embodiment, the cost function comprises a sum of distances traversed by j clients accessing n nodes to write k data objects. According to an embodiment, the sum of the distances is given as follows.

$$\sum_{j\in C}\sum_{n\in N}\sum_{k\in K} writes_{jk}\cdot dist_{jn}\cdot y_{jnk}$$

where a write variable $writes_{jk}$ indicates that a jth client writes a kth data object.

According to an alternative embodiment, the sum of the distances for retrievals is given as follows.

$$\sum_{j\in C}\sum_{n\in N}\sum_{k\in K} reads_{jk}\cdot dist_{jn}\cdot size_{k}\cdot y_{jnk}$$

where a size variable $size_k$ indicates a size of the kth data object.

According to an alternative embodiment, the cost function comprises a sum of storage costs for storing a kth data object on an nth node. According to an embodiment, the sum of the storage costs is given as follows.

$$\sum_{n\in N}\sum_{k\in K} sc_{nk}\cdot store_{nk}$$

where a storage cost variable sc_{nk} indicates a cost of storing the kth data object on the nth node. According to embodiments, the storage cost variable sc_{nk} indicates a size of the kth data object, a throughput of the nth node, or an indication that the kth data object resides at the nth node.

According to an alternative embodiment, the cost function comprises an access time, which indicates a most recent time that a kth data object was accessed on an nth node. According to another alternative embodiment, the cost function comprises a load time, which indicates a time of storage for a kth data object on an nth node. According to another alternative embodiment, the cost function comprises a hit ratio, which indicates a ratio of hits of transparent en route caches along a path from a jth client to an nth node.

The one or more placement constraints comprise a storage capacity constraint, a load capacity constraint, a node bandwidth capacity constraint, a link capacity constraint, a number of replicas constraint, a delay constraint, an availability constraint, or another placement constraint. According to an embodiment of the

method of instantiating the data placement heuristic, each of the placement constraints are categorized as an increasing constraint, a decreasing constraint, or a neutral constraint. The increasing constraints are violated by allocating too many of the k data objects. The decreasing constraints are violated by not allocating enough of the k data objects. The neutral constraints are not capable of being characterized as an increasing or decreasing constraints and can be violated in situation which allocate too many of the k data objects or too few of the k data objects.

The storage capacity constraint places an upper limit on a storage capacity for an nth node. The storage capacity constraint comprises an increasing constraint. According to an embodiment, the storage capacity constraint is given as follows.

$$\sum_{k \in K} size_k \cdot x_{nk} \leq SC_n \qquad \forall n$$

where a storage capacity variable SC_n indicates the storage capacity for the nth node.

The load capacity constraint places an upper limit on a rate of requests that an nth node can serve. The load capacity constraint comprises a neutral constraint. According to an embodiment, the load capacity constraint is given as follows.

$$\sum_{j \in C} \sum_{k \in K} reads_{jk} \cdot y_{jnk} \leq LC_n \qquad \forall n$$

where a load capacity variable LC_n indicates the load capacity for the nth node. According to an alternative embodiment, the load capacity constraint is given as follows.

$$\sum_{j \in C} \sum_{k \in K} (reads_{jk} + writes_{jk}) \cdot y_{jnk} \leq LC_n \qquad \forall n$$

The node bandwidth capacity constraint places an upper limit on a bandwidth for an nth node. The node bandwidth capacity constraint comprises a neutral constraint. According to an embodiment, the node bandwidth capacity constraint is given as follows.

$$\sum_{j \in C} \sum_{k \in K} reads_{jk} \cdot size_k \cdot y_{jnk} \leq BW_n \qquad \forall n$$

where a bandwidth capacity variable BW_n indicates the bandwidth for the nth node. According to an alternative embodiment, the bandwidth capacity constraint is given as follows.

$$\sum_{j \in C} \sum_{k \in K} (reads_{jk} + writes_{jk}) \cdot size_k \cdot y_{jnk} \leq BW_n \qquad \forall n$$

The link capacity constraint places an upper limit on a bandwidth between two nodes. The link capacity constraint comprises a neutral constraint. According to an embodiment, the link capacity constraint is given as follows.

$$\sum_{j \in C} \sum_{k \in K} reads_{jk} \cdot size_k \cdot z_{jlk} \leq CL_l \qquad \forall l$$

where an alternative access variable z_{jlk} indicates whether a jth client uses an lth link to access a kth data object and where link capacity variable CL_l indicates the bandwidth for the lth link. According to an alternative embodiment, the link capacity constraint is given as follows.

$$\sum_{j \in C} \sum_{k \in K} (reads_{jk} + writes_{jk}) \cdot size_k \cdot z_{jlk} \leq CL_l \qquad \forall l$$

The number of replicas constraint places an upper limit on the number of replicas. The number of replicas comprises an increasing constraint. According to an embodiment, the number of replicas constraint is given as follows.

$$\sum_{n \in N} x_{nk} \leq P \qquad \forall k$$

where a number of replicas variable P indicates the number of replicas.

The delay constraint places an upper limit on a response time for a jth client accessing a kth data object. The delay constraint comprises a decreasing constraint. The availability constraint places a lower limit on availability of the k data objects. The availability constraint comprises a decreasing constraint.

The metric scope comprises a client scope, a node scope, and an object scope. The client scope comprises the j clients considered by the data placement heuristic. The client scope ranges from local clients to global clients and includes regional clients, which comprise clients accessing a plurality of nodes within a region. The node scope comprises the n nodes considered by the data placement heuristic. The node scope ranges form a single node to all nodes and includes regional nodes. The object scope comprises the k data objects considered by the data placement heuristic. The object scope ranges from local objects (data objects stored on a local node) to global objects (all data objects stored within a distributed storage system) and includes regional objects.

The approximation technique places the k data objects with the goal of optimizing the cost function but without an assurance that the technique will provide an optimal cost value. According to embodiments, the approximation technique comprises a ranking technique, a threshold technique, an improvement technique, a

hierarchical technique, a multi-phase technique, a random technique, or another approximation technique. As discussed above, the terms "heuristic" and "approximation technique" in the context of the present invention have a broad meaning and apply to both heuristics and approximation algorithms.

The ranking technique begins with determining costs from the cost function for all combinations of clients, nodes, and objects within the metric scope. Next, the ranking technique sorts the costs according to ascending or descending values. The ranking technique then takes a first cost, which represent a *j*th client accessing a *k*th data object from an *n*th node and makes a decision to place the *k*th data object onto the *n*th node according to the one or more placement constraints. If a decreasing constraint or a neutral constraint is violated prior to placing the *k*th data object onto the *n*th node, the *k*th data object is placed onto the *n*th node. If an increasing constraint or a neutral constraint is not violated prior to placing the *k*th data object onto the *n*th node, the *k*th data object is placed onto the *n*th node. The ranking technique continues to consider placements according to the sorted costs until all of the combinations of clients, nodes, and objects within the metric scope have been considered.

An alternative of the ranking technique comprises a greedy ranking technique. The greedy ranking technique comprises the ranking technique plus an additional step of recomputing the costs of remaining items in the sorted list and sorting the remaining items according to the recomputed costs after each placement decision.

The threshold technique comprises the ranking technique with the additional step of limiting the sorted list to costs above or below a threshold. The random technique comprises randomly placing the *k* data objects onto the *n* nodes

The improvement technique comprises taking an initial placement of data objects on nodes and attempts to improve the initial placement by swapping placements of particular placements of objects on nodes. If the swapped placement provides a higher cost, the objects are returned to their previous placement. If an increasing constraint is violated with the swapped placement, the objects are returned to their previous placement. If a decreasing or neutral constraint was previously not violated but is violated with the swapped placement, the objects are returned to their previous placement. The improvement technique continues to swap object placements for a number of iterations.

The hierarchical technique comprises performing the ranking, threshold, or improvement technique at least twice where a following instance of the technique applies a broader metric scope. The multiphase technique comprises performing two of the approximation techniques in succession.

The evaluation interval comprises a measure of how often the method of instantiating the data placement heuristic is executed. According to an embodiment, the evaluation interval comprises a time period between executions of the data placement heuristic for one of the *n* nodes. According to another embodiment, the evaluation interval comprises a number of accesses by clients of a node such as every access or every tenth access.

The routing knowledge parameter comprises a specification for each of the *n* nodes regarding whether the node knows of the replicas stored on it or whether the node knows of all of the replicas stored within the distributed storage system or anything in between.

An embodiment of the method of instantiating the data placement heuristic is illustrated in figures 7A, 7B, and 7C as a flow chart. The method 700 begins in a first step 702 of receiving the cost function, a set of placement constraints, the metric scope, and a set of approximation techniques. According to an embodiment, the set of placement constraints comprises a single placement constraint. According to another embodiment, the set of placement constraints comprises a plurality of placement constraints. According to an embodiment, the set of approximation techniques comprise a single approximation technique. According to another embodiment, the set of approximation techniques comprise a plurality of approximation techniques.

The method continues in a second step 704 of determining a cost according to the cost function for each combination of *n* nodes and *k* data objects within the metric scope. A third step 706 comprises sorting the costs in ascending or descending order as appropriate for the cost function, which forms a queue.

In fourth or fifth steps, 708 or 710, the method 700 chooses the ranking technique or the threshold technique. According to an alternative embodiment, the method 700 chooses the random technique. According to another alternative embodiment, the method 600 chooses another approximation technique.

If the method 700 chooses the ranking technique, a seventh step 714 picks a placement of a *k*th data object on an *n*th node corresponding to a cost at a head of the queue. An eighth step 716 determines whether a neutral or decreasing constraint is

currently violated. If the neutral or decreasing constraint is currently not violated, a ninth step 718 determines whether a neutral or increasing constraint will not become violated by placing the *k*th data object on the *n*th node. If the eighth or ninth step, 716 or 718, provides an affirmative response, a tenth step 720 places the *k*th data object on the *n*th node. An eleventh step 722 determines whether the queue includes additional costs and, if so, the ranking technique continues.

The ranking technique continues in a twelfth step 724 of determining whether the ranking technique comprises a greedy technique. If so, a thirteenth step 726 recomputes the costs remaining in the queue and a fourteenth step 728 resorts the costs to reform the queue. The ranking technique then returns to the seventh step 714.

If the method 700 chooses the threshold technique, a fifteenth step 730 removes costs form the queue which do not meet a threshold. A sixteenth step 732 picks a placement of a *k*th data object on an *n*th node corresponding to the cost at a head of the queue. A seventeenth step 734 determines whether a neutral or decreasing constraint is currently violated. If the neutral or decreasing constraint is currently not violated, an eighteenth step 736 determines whether a neutral or increasing constraint will not become violated by placing the *k*th data object on the *n*th node. If the seventeenth or eighteenth step, 734 or 736, provides an affirmative response, a nineteenth step 738 places the *k*th data object on the *n*th node. A twentieth step 740 determines whether the queue includes additional costs and, if so, the threshold technique continues.

If the method 700 chooses the improvement technique, an initial placement of the *k* data objects on the *n* nodes within the metric scope has preferably been determined using the ranking or threshold technique. Alternatively, the initial placement of the *k* data objects on the *n* nodes within the metric scope is determined using the random technique. Alternatively, the initial placement of the *k* data objects on the *n* nodes within the metric scope is determined using another technique. Since the improvement technique begins with the initial placement of the *k* data objects placed on the *n* nodes, the improvement technique forms part of the multiphase technique where a first phase comprises the ranking, threshold, random, or other technique and where a second phase comprises the improvement technique.

In a twenty-first step 742, the improvement technique swaps a placement of two of the *k* data objects within the metric scope, which forms a swapped placement. A twenty-second step 744 determines whether the swapped placement incurs a worse

cost. A twenty-third step 746 determines whether the swapped placement violates an increasing constraint. A twenty-fourth step 748 determines whether a neutral or decreasing constraint is violated and whether the placement prior to swapping did not violate the neutral or decreasing constraint. If the twenty-first, twenty-second, or twenty-third step, 742, 744, or 746, provides an affirmative response, a twenty-fifth step 750 reverts the placement to the placement prior to swapping. A twenty-sixth step 752 determines whether to perform more iterations of the improvement technique. If so, the improvement technique returns to the twenty-first step 742.

In a twenty-seventh step 754, the method 700 determines whether to perform the hierarchical technique and, if so, the method 700 returns to the second step 704 with a broader metric scope. In a twenty-eighth step 756, the method 700 determines whether to perform the multiphase technique and, if so, the returns to the second step 704 to begin a next phase of the multiphase technique.

According to an embodiment, the method of instantiating the data placement heuristic along with the method of selecting the heuristic class forms the method of determining the data placement of the present invention.

An embodiment of the method of determining the data placement of the present invention is illustrated in figure 8 as a block diagram. The method 800 begins by inputting a workload, a system configuration, and a performance requirement to a first block 802, which select a heuristic class. A second block 804 receives the heuristic class and instantiates a data placement heuristic resulting in a placement of data objects on nodes of a distributed storage system. A third block 806 evaluates the data placement by applying a workload to the distributed storage system and measuring a performance and a replication cost, which are provided as outputs. According to an embodiment of the method 800, the outputs are provided to the first block 802, which begins an iteration of the method 800. In this embodiment, the method 800 functions as a control loop.

According to an embodiment of the method 800, the distributed storage system comprises an actual distributed storage system. In this embodiment, the method 800 functions as a component of the distributed storage system. According to another embodiment of the method 800, the distributed storage system comprises a simulation of a distributed storage system. According to this embodiment, the method 800 functions as a simulator. According to an embodiment that functions as the component of the actual distributed storage system, the outputs comprise an actual

workload, the performance, and the replication cost. According to an embodiment that functions as the simulator, the outputs comprise the performance and the replication cost. According to another embodiment that functions as the simulator, the outputs comprise the workload, the performance, and the replication cost. According to another embodiment that functions as the simulator, the outputs comprise the system configuration, the performance, and the replication cost.

According to an embodiment of the method 800, the first block 802 receives the inputs and selects the heuristic class. In an embodiment, the first block 802 provides the heuristic class to the second block 804 as a single parameter indicating the heuristic class. For example, the single parameter could indicate one of the heuristic classes identified in Table 3 (figure 8), such as storage constrained heuristics or local caching. In another embodiment, the first block 802 provides the heuristic class to the second block 804 as the heuristic parameters of the method of instantiating the data placement heuristic. In this embodiment, the first block 802 sets some of the heuristic parameters to defaults because the heuristic class does not specify these parameters. In an alternative of this embodiment, the first block 802 provides some of the heuristic parameters to the second block 804 and the second block 804 assigns defaults to the heuristic parameters not provided by the first block 802.

According to an embodiment of the method 800, the second block 804 instantiates the data placement heuristic for each evaluation interval within an execution of the second block 804. For example, if the evaluation interval is one hour and the execution is twenty four hours, the second block instantiates the data placement heuristic every hour for the twenty four hours. According to this example, the outputs from the third block 806 comprise the performance and the replication cost for twenty four instantiations of the data placement heuristic. According to another example, the evaluation interval is twenty-four hours and the execution is twenty-four hours. According to this example, the outputs from the third block 806 comprise the performance and the replication cost for a single instantiation of the data placement heuristic.

According to an embodiment of the method 800 that functions as the component of the distributed storage system and which operates as the control loop, a first operation of the control loop begins with the inputs comprising an anticipated workload, the system configuration, and the performance requirement. Second and

subsequent operations of the control loop use an actual workload, the performance, and the replication cost from the third block 806 to improve operation of the distributed storage system. According to an embodiment, the control loop improves the performance by tuning the heuristic parameters provided by the first block 802 to the second block 804. According to this embodiment, the heuristic parameters tuned by the first block 804 comprise previously provided heuristic parameters or previously provided defaults. According to another embodiment, the control loop improves the performance by keeping a history of actual workloads so that the first block 802 provides the heuristic parameters to the second block based upon time, such as by hour of day or day of week. According to this embodiment, the second block instantiates different data placement heuristics depending upon the time.

According to an embodiment of the method 800 that functions as the simulator and which operates as the control loop, a first operation of the control loop begins with the inputs comprising an initial workload, the system configuration, and the performance requirement. In this embodiment, the third block 806 outputs the workload, the performance, and the replication cost. Second and subsequent operations of the control loop vary the workload in order to identify heuristic parameters that instantiate a data placement heuristic that operates well under a range of workloads.

According to another embodiment of the method 800 that functions as the simulator and which operates as the control loop, a first operation of the control loop begins with inputs comprising the workload, an initial system configuration, and the performance requirement. In this embodiment, the third block 806 outputs the system configuration, the performance, and the replication cost. Second and subsequent operations of the control loop vary the system configuration in order to identify a particular system configuration that operates well under the workload.

According to another embodiment of the method 800 that functions as the simulator and which operates as the control loop, a first operation of the control loop begins with inputs comprising an initial workload, an initial system configuration, and the performance requirement. In this embodiment, the third block outputs the workload, the system configuration, the performance, and the replication cost. Second and subsequent operations of the control loop vary the workload or the system configuration in order to identify a particular system configuration and a data placement heuristic that operates well under a range of workloads.

The foregoing detailed description of the present invention is provided for the purposes of illustration and is not intended to be exhaustive or to limit the invention to the embodiments disclosed. Accordingly, the scope of the present invention is defined by the appended claims.